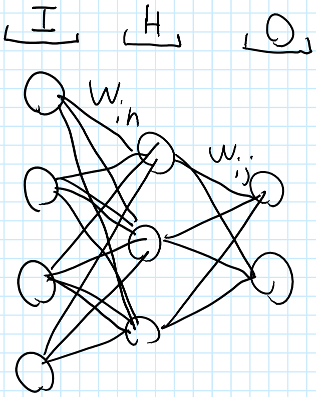
Neural Network Research

Thursday, December 26, 2013   12:51 PM

Feedforward Network



1. Inputs take in some values, and feed them forward to the output layer.
   - Feed forward occurs w/ a sigmoid activation such as

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

   - ~~For some node $j \in (H_n \cup O)$~~

$$\cancel{O_j = \frac{1}{1+\exp\left(-\sum_{i \in H_n} O_i w_{ij}\right)}}$$

$\sigma(x) = \tanh$

2. Finally some output is reached such that an error can be defined.

3 Feed Forward Activation

$$O_i = f_i\left(\sum_{j \in A_i} w_{ij} O_j\right)$$

http://stackoverflow.com/questions/14507194/activation-function-for-multilayer-perceptron

- For every layer except for i=0

Error Backpropagation

1. Definitions
   - node Error: $-\frac{\partial E}{\partial net_j} = \delta_j$
   - the weight gradient $= \Delta w_{ij} = -\frac{\partial E}{w_{ij}}$
   - the set of anterior nodes to node i: $\boxed{A_i} \{j : \exists w_{ij}\}$
   - the set of posterior nodes to node i $= \boxed{P_i} \{i : \exists w_{ij}\}$

2. The gradient

First off, we need to calculate the ∇ for all of the weights against some error function.

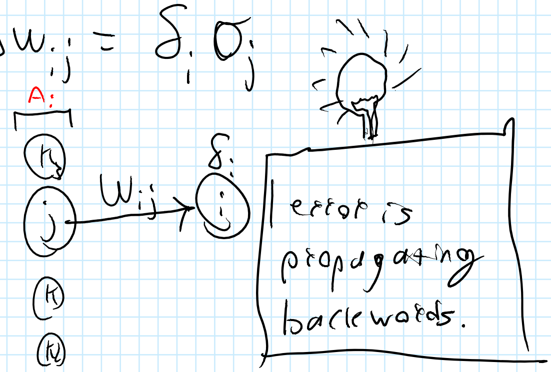
$$\Delta w_{ij} = -\frac{\partial E}{\partial w_{ij}} = -\frac{\partial E}{\partial net_i}\frac{\partial net_i}{\partial w_{ij}}$$

We can then define this chain rule further.

$$\frac{\partial net_i}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_{k \in \boxed{A_i}} W_{ik} O_k = O_j$$

therefore,

$$\Delta w_{ij} = \delta_i O_j$$



4. Calculating Output Error

$$E = \frac{1}{2}\sum_{o \in O}(t_o - O_o)^2$$

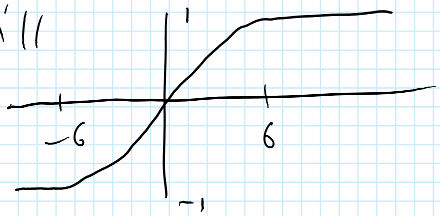
- recalling that node delta is

$j \in A_i$ multilayer perceptron

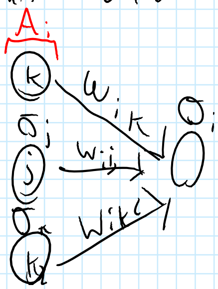
- For every layer except for $i=0$ tanh defines the sigmoid activation function $f_i(n) = \tanh(n)$
- This function is antisemetric

~~tan~~ ll

ta



- This also appears as follows



- recalling that node delta is

$$\delta_i = -\frac{\partial E}{\partial net_i}$$

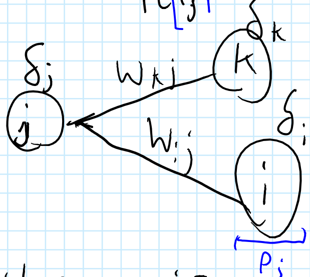
we can derive that

$$\delta_o = t_o - \sigma_o$$

Error Backpropigation

For a hidden node, we must propagate the error backwords:

$$\delta_j = -\sum_{i \in P_j} \frac{\partial E}{\partial net_i} \frac{\partial net_i}{\partial y_j} \frac{\partial y_j}{\partial net_j}$$



$$\delta_i = -\frac{\partial E}{\partial net_i} = -\sum_{a \in P_i} \frac{\partial E}{\partial net_a} \frac{\partial net_a}{\partial y_i} \frac{\partial y_i}{\partial net_i}$$

remember $-\frac{\partial E}{\partial net_a} = \delta_a$

When j is a node before the output layer, we can define

$$\delta_j = \left(\sum_o \delta_o w_{jo}\right) \sigma_j'(net_j)$$

We can then generalize this to any hidden node:

$$\boxed{\delta_j = \sigma'(net_j) \sum_{i \in P_j} \delta_i w_{ij}}$$

for implementation:

$$\delta_j = (1 - (\sigma(net_j))^2) \sum \delta_i w_{ij}$$

Steps:

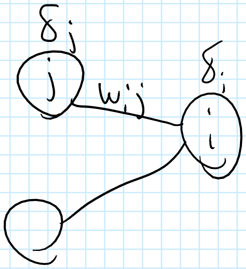
for implementation:

$$\boxed{\delta_j = (1-(\sigma(net_j))^2)\sum_{i\in P_j} \delta_i w_{ij}}$$

Bringing back the original model

$$\Delta w_{ij} = \delta_i \bar{O}_j$$

We can contextualized the node error derivation.



$$w_{ij} = \Delta w_{ij} + w_{ij}$$
$$= \delta_i \cdot \bar{O}_j + w_{ij}$$
$$= \text{error} \cdot \text{Output} + \text{last weight.}$$

Neural Network Implementation

Thursday, December 26, 2013 3:12 PM

For some undefined dataset the implementation for a neural network using error-backpropagation.

1. **Classes**
   - **a.** Neuron
     - **i.** Error
       - **1)** Output Neurons

$$\delta_o = t_o - \sigma_o(net_o)$$

       - **2)** Hidden Neurons

$$\delta_j = \sigma'(net_j)\sum_{i \in P_j} \delta_i w_{ij} = (1 - \sigma^2(net_j))\sum_{i \in P_j} \delta_i w_{ij}$$

     - **ii.** Output
       - **1)** Input

$$O_{in} = input \text{ ; no sigmoid applied}$$

       - **2)** Hidden/Output Neurons

$$O_i(net) = \tanh(net_i)$$

       - 3) BIAS

$$O_b = 1$$

     - **iii.** Net

$$net_i = \sum_{j \in A_i} w_{ij} O_j$$

   - **b.** Neural Network
     - i. Error

$$E = \frac{1}{2}\sum_{o \in O}(t_o - \sigma_o)^2$$

     - ii. Algorithm
       1) Set the weights throughout the entire neural network to random values bounded by [-1, 1]
       2) Begin the training using the *train(DataSet)* function which repeats the following methods until reaches a certain threshold
          - a) For a given training set, feed forward the inputs X_i through the network using the *feedforward(double[])* function.
          - b) Once the inputs are fed forward, calculate the global error for the ith training set using the equation depicted above.
          - c) Using the given global error, backpropagate that error using the *backpropagate(double[])* function.
          - d) Finally, after node deltas have been calculated, run the *updateweights()* function.

   - c. Weight
     - i. Update Weight Rule

$$\Delta w_{ij} = -\frac{\partial E}{\partial w_{ij}} = -\frac{\partial E}{\partial net_i}\frac{\partial net_i}{\partial w_{ij}}$$

$$= \delta_i \cdot \sigma_j$$